Xilinx Answer 65062
AXI Memory Mapped for PCI Express Address Mapping

Important Note: This downloadable PDF of an Answer Record is provided to enhance its usability and readability. It is important to note that Answer Records are Web-based content that are frequently updated as new information becomes available. You are reminded to visit the Xilinx Technical Support Website and review (Xilinx Answer 65062) for the latest version of this Answer.

Introduction

This document provides a conceptual explanation on how the address translation between the AXI domain and PCIe domain and vice versa is done in the AXI Memory Mapped for PCI Express core. There is no additional information in this document than what has already been provided in the core product guide (PG055). Readers are encouraged to go through the product guide for the comprehensive details on the address translation mechanism and the overall functionality of the core.

What are BARs?

BARs, or Base Address Registers, are the starting address of a contiguous mapped address in system memory or I/O space. In PCIe, an Endpoint requests a size of contiguous memory (or I/O space), which is then mapped by the upstream device's memory manager, and the Base Address Register is programmed with the base address for that Endpoint's BARx field in the endpoint's configuration space. For example, a 32-bit BAR0 is offset 10h in PCI Compatible Configuration Space – and post enumeration would contain the start address of BAR0.

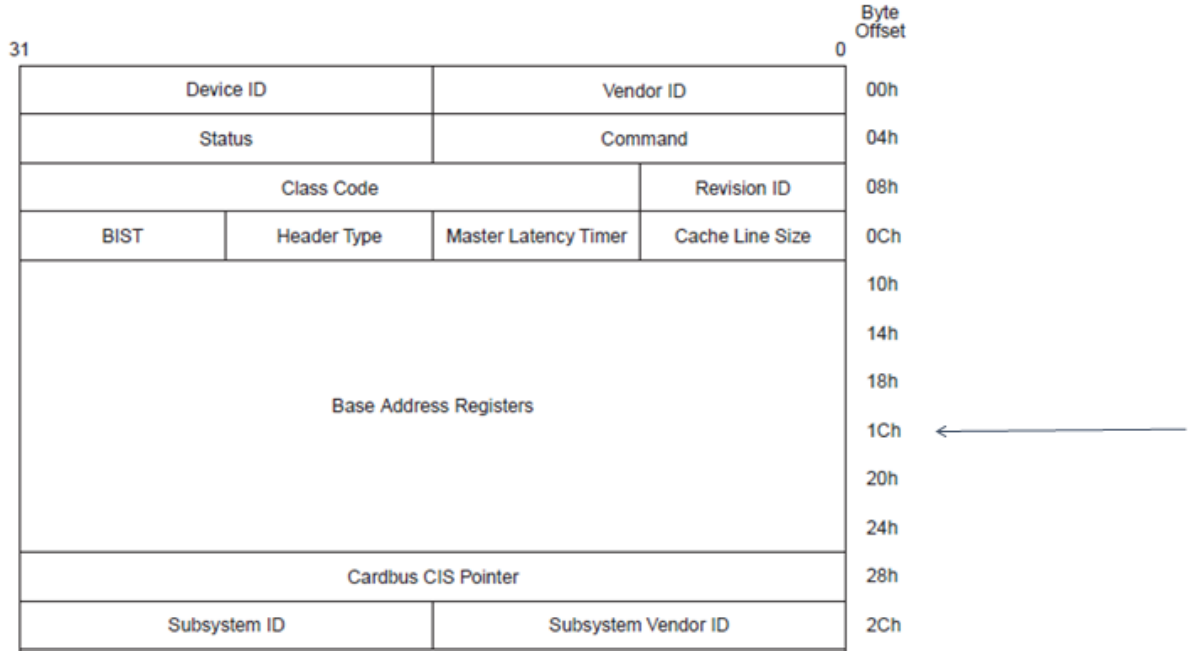


Figure 1 - PCIe PCI Compatible Configuration Space for Endpoint (Type0) - Shows space for 6 32-bit BAR or 3 64-bit BAR

As a Root Port in PCIe, this is the space that you are requesting from your own memory manager, to be used for your driver operations, etc. It should be noted that the Endpoint's BAR and the Root Port's BAR do not overlap (i.e. the Endpoint's BAR is not contained in the same the space as the Root Port's BAR).

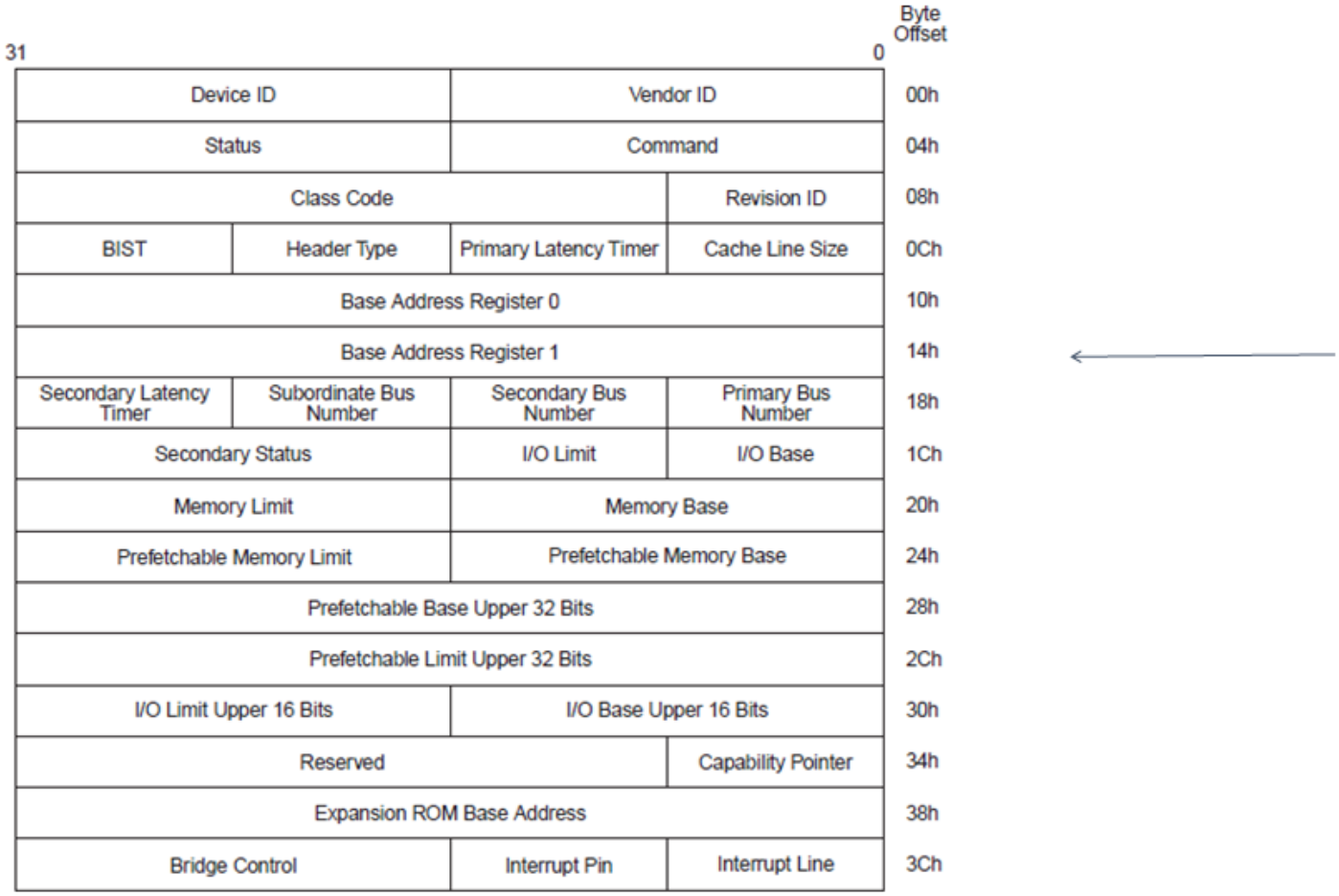


Figure 2 - PCIe PCI Compatible Configuration Space for Root Port (Type 1) - Shows space for 2 32-bit BAR or 1 64-bit BAR

For any other peripheral device to write data to the PCIe Endpoint in question, or read data from the PCIe Endpoint, that request would be submitted at the Root Port with an address within one of the Endpoint's BARs as shown in Figure 3.

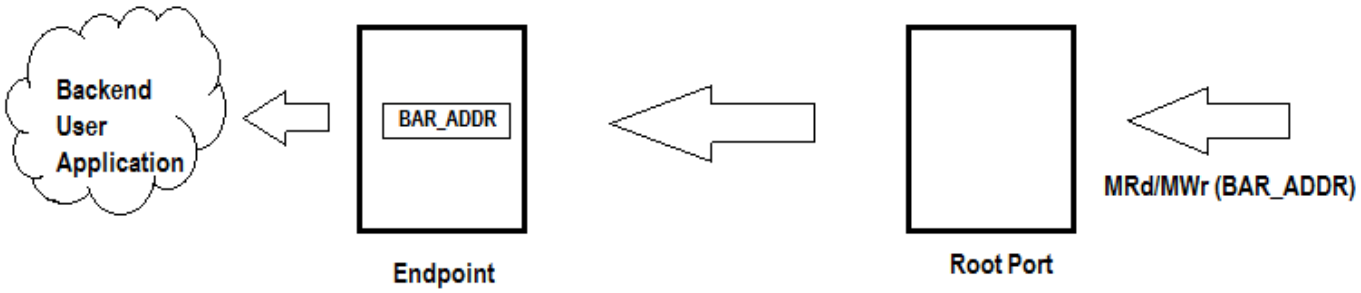


Figure 3 – Address Read/Write from Rootport to Endpoint

In an AXI system, particularly one with its own Embedded Microprocessor (Figure 4), the memory available within that AXI system is mapped by a set of addresses determined by the user when configuring the design in the Address Editor (Figure 5).

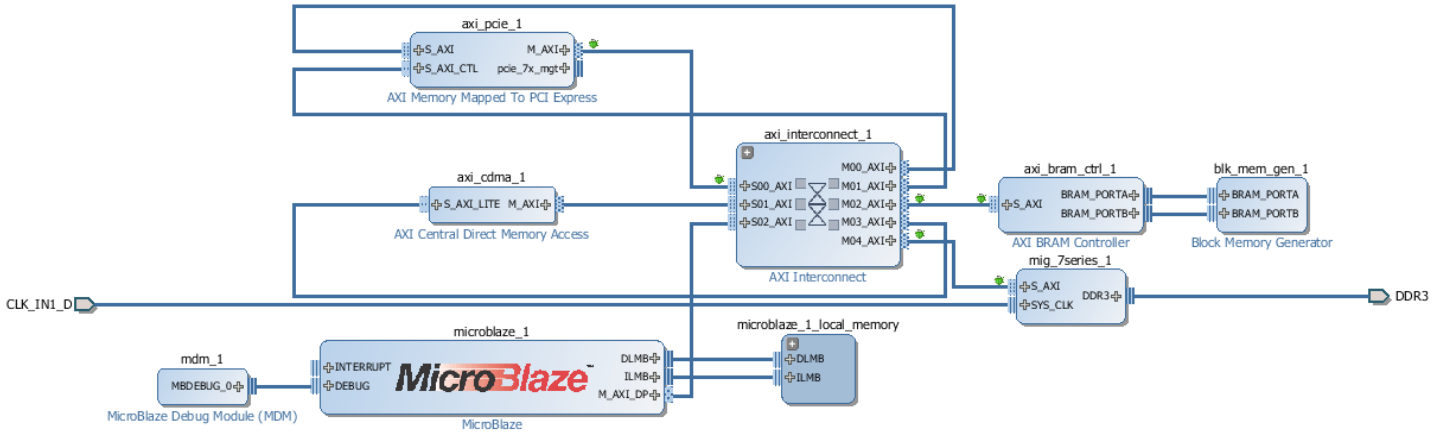


Figure 4 - AXI System with AXI PCIe and Microblaze

Diagram | Address Editor

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
axi_cdma_1					
Data (32 address bits : 4G)					
axi_cdma_1	S_AXI_LITE	Reg	0xD000_0000	64K	0xD000_FFFF
axi_bram_ctrl_1	S_AXI	Mem0	0xC000_0000	32K	0xC000_7FFF
axi_pcie_1	S_AXI_CTL	CTL0	0x7600_0000	64K	0x7600_FFFF
axi_pcie_1	S_AXI	BAR0	0x7601_0000	64K	0x7601_FFFF
mig_7series_1	S_AXI	memaddr	0x8000_0000	1G	0xBFFF_FFFF
axi_pcie_1					
M_AXI (32 address bits : 4G)					
axi_pcie_1	S_AXI_CTL	CTL0	0x7600_0000	64K	0x7600_FFFF
axi_pcie_1	S_AXI	BAR0	0x7601_0000	64K	0x7601_FFFF
axi_cdma_1	S_AXI_LITE	Reg	0xD000_0000	64K	0xD000_FFFF
axi_bram_ctrl_1	S_AXI	Mem0	0xC000_0000	32K	0xC000_7FFF
mig_7series_1	S_AXI	memaddr	0x8000_0000	1G	0xBFFF_FFFF
microblaze_1					
Data (32 address bits : 4G)					
microblaze_1_local_memory/dlmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	8K	0x0000_1FFF
axi_bram_ctrl_1	S_AXI	Mem0	0xC000_0000	32K	0xC000_7FFF
axi_pcie_1	S_AXI_CTL	CTL0	0x7600_0000	64K	0x7600_FFFF
axi_pcie_1	S_AXI	BAR0	0x7601_0000	64K	0x7601_FFFF
mig_7series_1	S_AXI	memaddr	0x8000_0000	1G	0xBFFF_FFFF
axi_cdma_1	S_AXI_LITE	Reg	0xD000_0000	64K	0xD000_FFFF
Instruction (32 address bits : 4G)					
microblaze_1_local_memory/ilmb_bram_if_cntlr	SLMB	Mem	0x0000_0000	8K	0x0000_1FFF

Figure 5 - Address Editor

AXI BAR and PCIe BAR

There is a set of memory spaces and addressing for both types of interconnects to the core. On one side, you have the locations on the main system's Memory (typically in DRAM). For an endpoint this is assigned through the Root Port, from

the system's memory management interface. The other side of the core is an AXI interconnect – typically with a DMA engine controlling data movement to BRAM or the MIG controller, and has independent addressing from the system level.

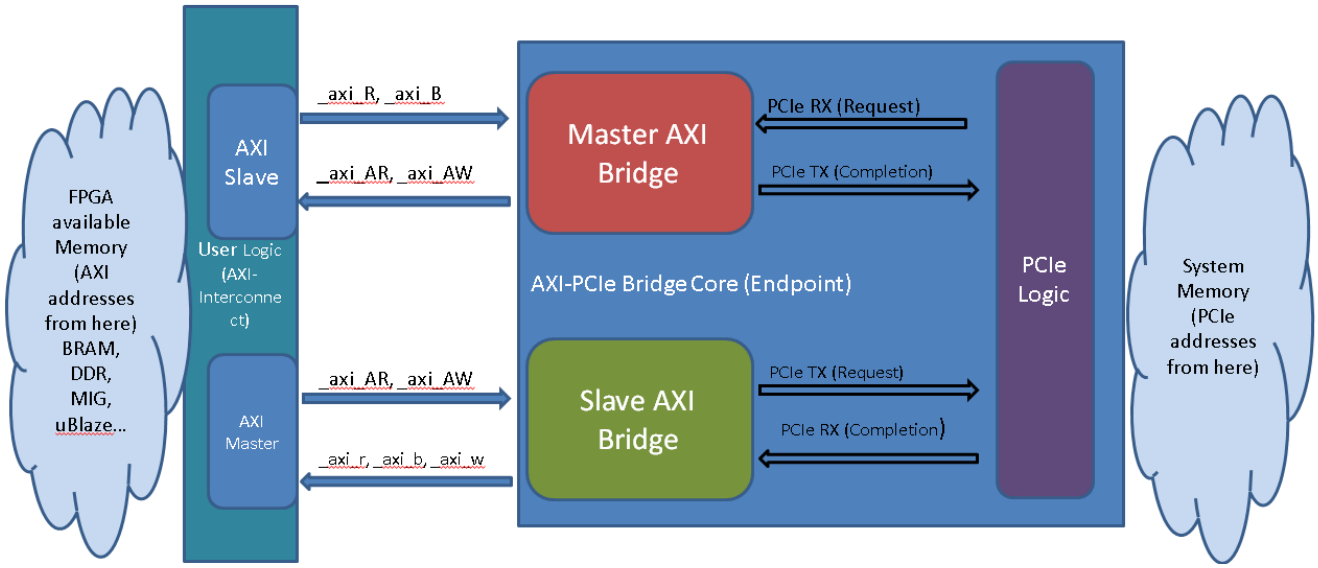


Figure 6 - Conceptual AXI PCIe Bridge Connections

The goal of the GUI configuration is to define at least the size of the BARs on both sides, and if possible map the addresses of one side to the other, prior to being up in a system.

PCIe BAR Configuration and PCIe to AXI Address Translation Parameter

Figure 7 is used for 2 purposes. The first is to define how many PCIe registered spaces are requested for the endpoint, the type (typically memory) and the size. The second is where the user will define the parameter **C_PCIEBAR2AXIBAR**. This parameter allows the core to take an incoming request from the link partner, with a specific address and size, and internally translate that to the AXI Address and length that would be recognized by the system on the other end of the core.

In the example shown, it enables two BARs: BAR0 and BAR1 of 16KB each. At enumeration time in PCIe, the Root Port will enumerate both of these BARs with their respective base addresses in system memory (contiguously mapped).

It is expected that all memory requests made to the endpoint will now be addressed within the two address ranges defined by BAR0 and BAR1.

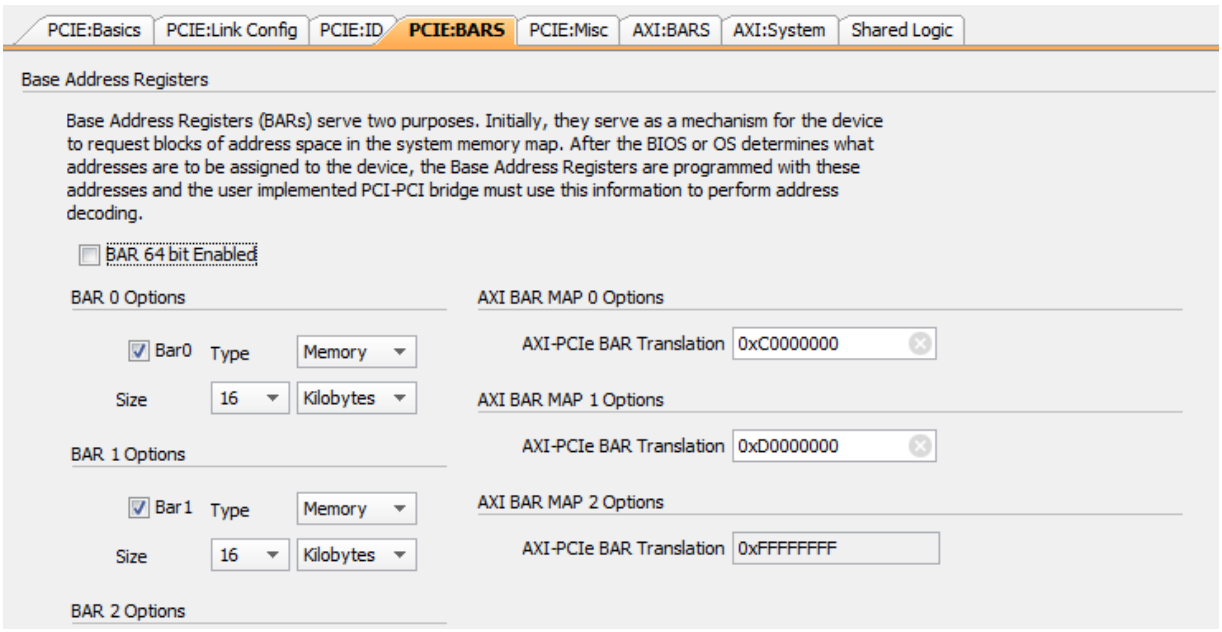


Figure 7 - Core Configuration PCIE:BARS – AXI PCIe Core Master Bridge Memory Map

There is one address translation parameter for each BAR. Three MAP options on the right, shown in the example are mapped to three BARs, on the left, respectively. Figure 8 illustrates how the address translation parameters and the BARs are mapped to the devices on the AXI system. BAR0 is mapped to AXI BRAM controller and BAR1 is mapped to AXI CDMA. Any requests made to the 16KB range, from the link partner, with the base address defined in BAR0 will be addressed to the AXI BRAM controller and similarly to the AXI CDMA in the case of BAR1.

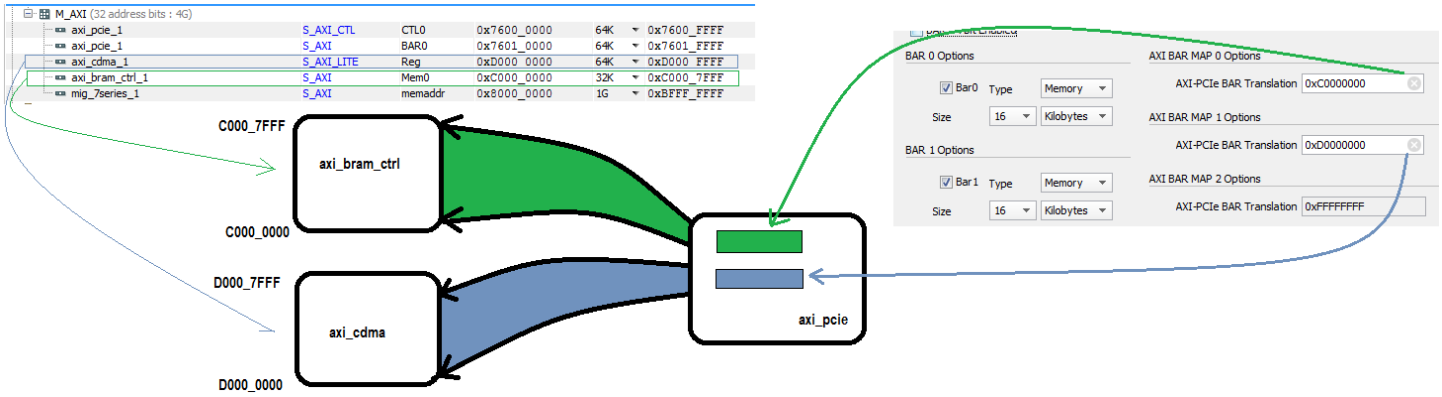


Figure 8 – Address Translation Parameter Mapping with AXI Devices

AXI to PCIe Address Translation Parameter

Figure 10 shows the AXI:BARS tab of the core configuration GUI. The AXI BAR size and Base Address (and high address) are defined within the AXI system – address Editor shown in Figure 9.

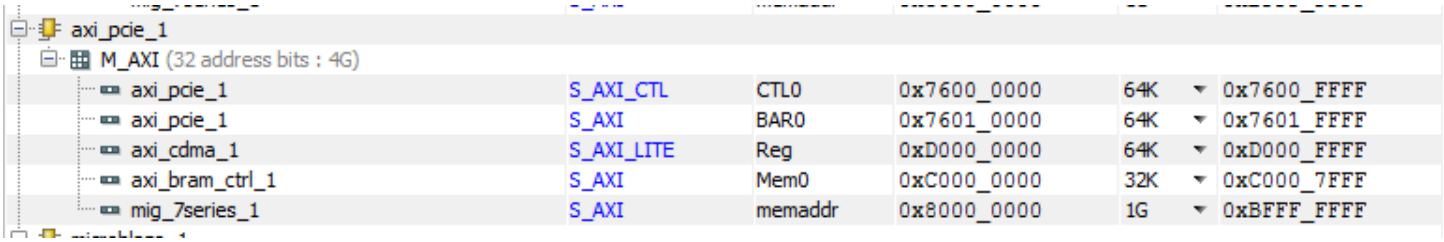


Figure 9 - Address Editor

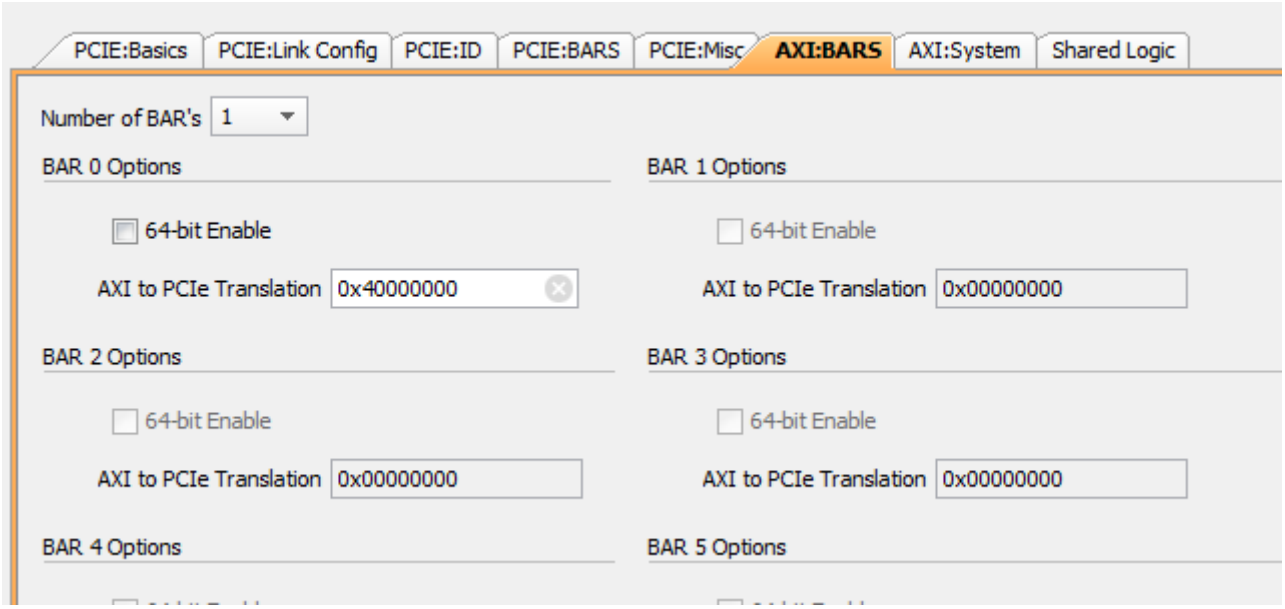


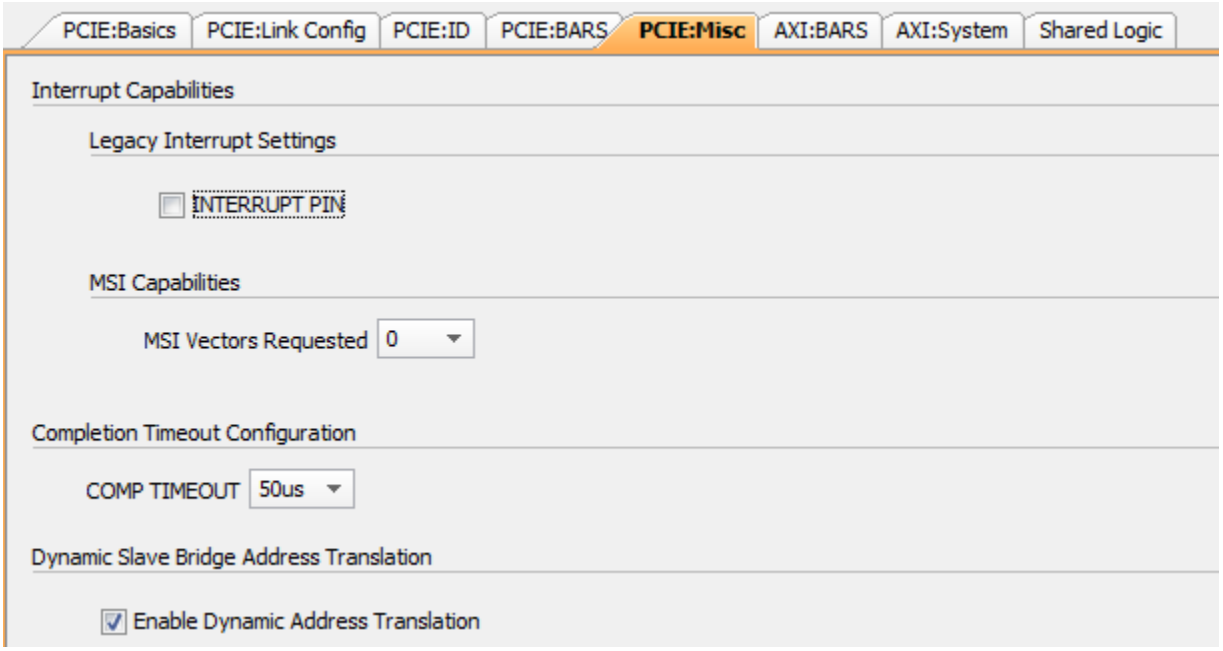
Figure 10 - AXI-PCIe Bridge Configuration GUI - AXI:BARS

The boxes on this tab set the core parameter **C_AXIBAR2PCIEBAR**. A user would fill in the data for this tab at core configuration time if:

1. The User plans on the endpoint sending MemRd/MemWr requests upstream (including Interrupt MemWr), and receiving completions to those requests.
2. The User knows ahead of time exactly which addresses the Endpoint would be instructed to write to in the system memory (at the link partner).

If (1) is False, then this tab data can be left as default. If (1) is true, but (2) is false (which will be the case for the majority of open systems), then the user needs to take 2 actions:

1) On the PCIE:Misc tab, check the 'Enable Dynamic Address Translation' in the Dynamic Slave Bridge Address box. This ensures that the **C_AXIBAR2PCIEBAR_x** can be edited in flight once the PCIe bus has been initialized and the PCIe side has been enumerated.



2) Once the system is up and running, the OS/drivers of the endpoint will get the correct address for MemRd / MemWr requests initiated by the core, and transmit this to a desired location (via PCIe) on the Endpoint. At that time, the user should write to the **C_AXIBAR2PCIEBAR_x** with that address (or the address offset), which will then allow the user to put in an AXI address mapped to the initiation of requests (and receipt of completions w/data), and have that translated to the PCIe address dedicated to requests specifically from that Endpoint.

Figure 11 illustrates the AXI to PCIe address translation in the case where the endpoint knows the system memory address before the enumeration of the PCIe devices by the host.

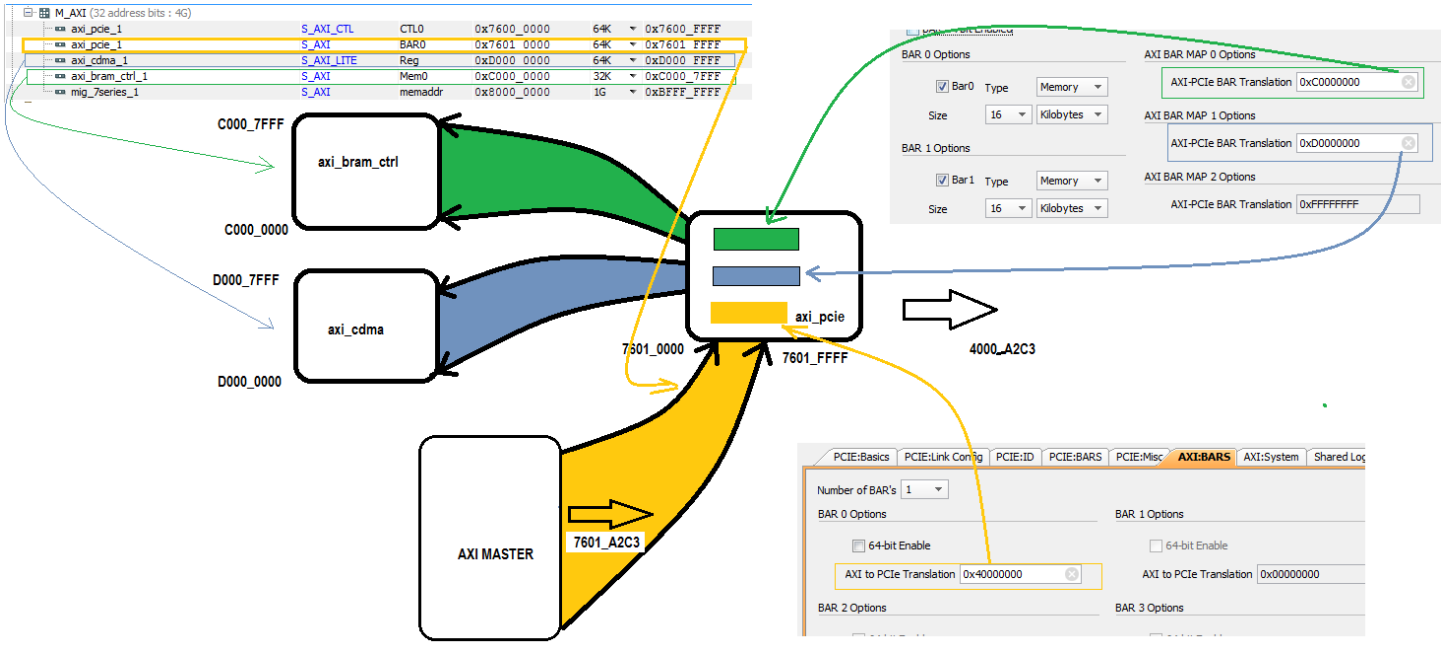


Figure 11 - AXI to PCIe Address Translation

Appendix A: Address Translation Examples

The examples below are taken from PG055. For more examples, please refer to Address Translation section of PG055.

Translate 32-bit AXI address to a 32-bit address for PCIe

This example shows the generic settings to set up three independent 32-bit AXI BARs and address translation of AXI addresses to a remote 32-bit address space for PCIe. This setting of AXI BARs does not depend on the BARs for PCIe within the AXI Bridge for PCI Express core.

In this example, where C_AXIBAR_NUM=1:

C_AXIBAR_AS_0=0
C_AXIBAR_0=0x12340000
C_AXI_HIGHADDR_0=0x1234FFFF
C_AXIBAR2PCIEBAR_0=0x5671XXXX (Bits 15-0 do not matter as the lower 16-bits hold the actual lower 16-bits of the PCIe address)

- Accessing the Bridge AXIBAR_0 with address 0x12340ABC on the AXI bus yields 0x56710ABC on the bus for PCIe.

Translate 32-bit PCIe address to a 32-bit address for AXI

This example shows the generic settings to set a BAR for PCIe and address translation of addresses for PCIe to a remote AXI address space. This setting of BARs for PCIe does not depend on the AXI BARs within the bridge.

In this example, where C_PCIEBAR_NUM=1, the following range assignments are made:

BAR 0 is set to 0x20000000_ABCD8000 by the Root Port:

C_PCIEBAR_LEN_0=15
C_PCIEBAR2AXIBAR_0=0x1234_0XXX (Bits 14-0 do not matter)

• Accessing the Bridge PCIEBAR_0 with address 0x20000000_ABCDFFF4 on the bus for PCIe yields 0x1234_7FF4 on the AXI bus.

Appendix B: AXI Base Address Translation Configuration Registers (Ref: PG055)

The screen capture below is from PG055. It has been presented here for a quick reference.

AXI Base Address Translation Configuration Registers (Offset 0x208 - 0x234)

The AXI Base Address Translation Configuration Registers and their offsets are shown in Table 2-24 and the register bits are described in Table 2-25. This set of registers can be used in two configurations based on the top-level parameter C_AXIBAR_AS_n. When the BAR is set to a 32-bit address space, then the translation vector should be placed into the AXIBAR2PCIEBAR_nL register where n is the BAR number. When the BAR is set to a 64-bit address space, then the most significant 32 bits are written into the AXIBAR2PCIEBAR_nU and the least significant 32 bits are written into AXIBAR2PCIEBAR_nL. These registers are only included if C_INCLUDE_BAR_OFFSET_REG = 1.

Table 2-24: **AXI Base Address Translation Configuration Registers**

Offset	Bits	Register Mnemonic
0x208	31-0	AXIBAR2PCIEBAR_0U
0x20C	31-0	AXIBAR2PCIEBAR_0L
0x210	31-0	AXIBAR2PCIEBAR_1U
0x214	31-0	AXIBAR2PCIEBAR_1L
0x218	31-0	AXIBAR2PCIEBAR_2U
0x21C	31-0	AXIBAR2PCIEBAR_2L
0x220	31-0	AXIBAR2PCIEBAR_3U
0x224	31-0	AXIBAR2PCIEBAR_3L
0x228	31-0	AXIBAR2PCIEBAR_4U
0x22C	31-0	AXIBAR2PCIEBAR_4L
0x230	31-0	AXIBAR2PCIEBAR_5U
0x234	31-0	AXIBAR2PCIEBAR_5L

Appendix C: Address Translation Core Parameters

The screen captures on core parameters below have been provided here for a quick reference. For more details, please refer to PG055.

C_AXIBAR_NUM	Number of AXI address apertures that can be accessed	1: BAR_0 enabled 2: BAR_0, BAR_1 enabled 3: BAR_0, BAR_1, BAR_2 enabled 4: BAR_0 through BAR_3 enabled 5: BAR_0 through BAR_4 enabled 6: BAR_0 through BAR_5 enabled	6	Integer
C_AXIBAR_0	AXI BAR_0 aperture low address	Valid AXI address[(1)(3)(4)]	0xFFFF_FFFF	std_logic_vector
C_AXIBAR_HIGHADDR_0	AXI BAR_0 aperture high address	Valid AXI address[(1)(3)(4)]	0x0000_0000	std_logic_vector
C_AXIBAR_AS_0	AXI BAR_0 address size	0: 32 bit 1: 64 bit	0	Integer
C_AXIBAR2PCIEBAR_0	Initial address translation from an AXI BAR_0 address to a PCI Express address	Valid address for PCIe[(2)]	0xFFFF_FFFF	std_logic_vector
C_PCIEBAR_NUM	Number of address for PCIe apertures that can be accessed	1: BAR_0 enabled 2: BAR_0, BAR_1 enabled 3: BAR_0, BAR_1, BAR_2 enabled	3	Integer
C_PCIEBAR_LEN_0	Specifies the size of the PCIe BAR	13-31	16	Integer
C_PCIEBAR2AXIBAR_0	Initial address translation from an AXI BAR_0 address to a PCI Express address	Valid AXI address	0x0000_0000	std_logic_vector

C_PCIEBAR_AS	Configures PCIEBAR aperture width to be 32 bits wide or 64 bits wide	0: Generates three 32-bit PCIEBAR address apertures. 32-bit BAR example: PCIEBAR_0 is 32 bits PCIEBAR_1 is 32 bits PCIEBAR_2 is 32 bits 1: Generates three 64 bit PCIEBAR address apertures. 64-bit BAR example: PCIEBAR_0 and PCIEBAR_1 concatenate to comprise 64-bit PCIEBAR_0. PCIEBAR_2 and PCIEBAR_3 concatenate to comprise 64-bit PCIEBAR_1. PCIEBAR_4 and PCIEBAR_5 concatenate to comprise 64-bit PCIEBAR_2	1	Integer

Revision History

07/25/2015 - Initial release